A short introduction to using Nav2000

Ola Ringdahl, ringdahl@cs.umu.se

May 14, 2008

1 Introduction

This guide explains the main details about the Nav2000 system from a user perspective. Nav2000 is a mobile-robot middleware implemented in Java, allowing efficient configuration of the robot's sensors and actuators. In many ways it has the same functionality as ARIA, the software shipped with the AmigoBot and Pioneer robots from MobileRobots. The main difference is that Nav2000 is more general and works better in Java than ARIA's Java API does. The Nav2000 system can be run on several different computers if some software modules require more computing power. It supports many different sensors and it is fairly easy to add new sensors as well. A dedicated health monitor system keeps track of all software modules loaded onto the local computer, and also communicates with health monitors in all other computers running the system. The overall health of every module as well as a more detailed description of possible problems is presented graphically. In addition to this, the system uses logfiles to enable debugging and performance analysis of hardware and software modules. Currently the system supports four different robot platforms; a forest machine, a 3D simulator, the AmigoBot and Pioneer robots. This guide focus primarily on how to use the system for the two latter ones. The system was developed as part of the project Autonomous navigation for forest machines [3] and was primarily designed and implemented by Thomas Johansson. The work has been published in [1, 2].

2 Ini and cfg-files

Each robot has its own configuration (cfg) and ini-files located in the configuration directory *...\cm_ini\<vehicleName>* (e.g c:\...\cm_ini\Pioneer\).

Which robot and sensors to use are specified in a cfg-file under this directory (e.g. *amigobot.cfg*). Several cfg-files can be made for different configurations of the same robot. The following are some examples of the commands you can give in the cfg-file:

- # is a comment. You may not insert comments after a command (i.e. it must be either above or below that row).
- *register.default.localnode = 192.168.1.1* // the IP-address of the local computer[1].

[1]Note that this is the IP-address of the computer running Nav2000, not the robot's IP.

- *debug = INFORMATION+EXCEPTION* // how much debug data will be stored in the log-file. ALL will log all data. Each class specifies what will be logged by each command. (logfiles are stored in the \log directory, e.g c:\\...\\cm_ini\\Pioneer\\log\\). For all available commands, see com.ifor.utils.Logger.

- *load.amigobot = com.ifor.vehicle.Pioneer* // the Class *Pioneer* in the package *com.ifor.vehicle* will be loaded, and its ini-file is named *amigobot.ini.* This module will be also be recognized by the Registry by the name "amigobot".

- *alias.vehicle = amigobot* // The amigobot is now also known as "vehicle" (could be used for writing more general code for example).

Each sensor and robot has an ini-file located in the sub-directory *ini*\\ (e.g c:\\...\\cm_ini\\Pioneer\\ini\\) where it can be configured. Examples of things that can be configured here are:

- All sensors have a *mounting pose* (x, y ,z, roll, pitch, yaw) that specifies where on the robot it is mounted relative to the robot position (normally the center of the robot).

- For a sonar array, the mounting pose of each sonar is normally specified relative to the mounting pose of the array (e.g the 6 sonars on the front of the AmigoBot is regarded as a sensor; a *RangeArraysensor* of 6 sonar sensors). The max range of each sonar can be specified here to.

- The AmigoBot (and some sensors) has an IP-address that can be specified in its ini-file.

A graphical interface is also available for configuring which modules to load (the cfg-file) and also each modules properties (the ini-file). It is called *ConfigurationManager* and is located in the package com.ifor.config.

3 Java classes

The system is started by *Nav2000.java* (located in com.ifor.main). The cfg-file to use is given as an argument to the Nav2000 constructor. *Nav2000.open()* and *.close()* opens and closes all objects loaded (sensors, actuators and robots).

Nav2000 will create Objects of all classes specified in the cfg-file. To get a reference to a specific object, *Registry.java* is used (located in com.ifor.utils). The static method *lookUp(String name)* returns a reference to the Object corresponding to *name* (the same as in the cfg-file and the name on the ini-file, e.g *pioneerpositionsensor*).

Vehicle.java: The system has two modes of operation: *Auto* and *Tele.* Tele-mode is used for tele-operation and is activated by the method *setTele().* In this mode, *setThrottle()* and *setTurn()* is used to set a throttle between 0 and 1 (a fraction of max speed) and a turning rate between 0 and 1 (a fraction of max turning rate). In Auto- mode, *setSpeed(speed)* and *setAngle(angle)* is used instead. The Pioneer and AmigoBot robots implements one more method; *setSpeed(leftSpeed, rightSpeed)*, which sets individual wheel speeds. Speed is in m/s, heading (yaw) and steering angle are in radians. TimeStamps are in ms.

To get the robot's position, heading, speed and steering angle, use the appropriate sensor:

- PositionSensor - getPosition() returns a Position object (contains getX, getY, getZ and getTimeStamp) $[m]$
- Headingsensor - getHeading() returns a TimeStampedDouble object (contains getValue and getTimeStamp) $[rad]$
- AttitudeSensor - getAttitude() returns an Attitude object (roll, pitch, yaw, timeStamp). Yaw is the heading. $[rad]$
- SpeedSensor - getSpeed() returns a TimeStampedDouble object (getValue and getTimeStamp) $[m/s]$
- AngleSensor - getAngle() returns a TimeStampedDouble object (getValue and getTimeStamp) $[rad]$
- RangeArraySensor - a sensor that delivers one or more ranges (in meters) to obstacles (e.g PioneerSonarArray). The following methods are used:
  - *Pose[] getPoses()* - return an array of mounting poses, relative to the sensor group ref point (i.e in which direction each sonar points and its mounting position relative to the array as specified in its ini-file)
  - *double[] getRanges()* - return an array of range readings, in meters

In addition to polling (the get-methods), all sensors also supports listeners (observer pattern). To use this, an inner class and a couple of methods must be added. The following code explains how to listen to a speed sensor:

```
---------------------------------------------------
SpeedSensor speedSensor = ...
// create a new observer
SpeedObserver speedObserver = new SpeedObserver();
// add an observer to the speed sensor
speedSensor.addSpeedObserver(speedObserver);

// Inner class for listening to speed
class SpeedObserver implements Observer {
   public void update(Observable o, Object arg) {
      speed = (TimeStampedDouble) arg;
      calcSomething();
   }
}
public void calcSomething() {
```

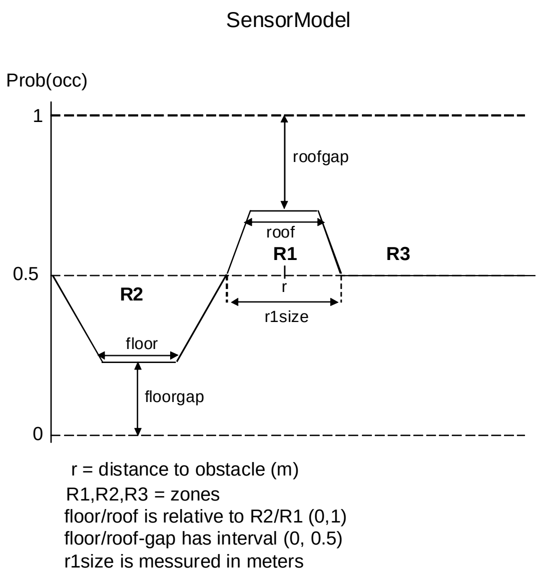

SensorModel
Prob(occ)
1
0.5
0
roofgap
roof
R1
R3
R2
r
r1size
floor
floorgap
r = distance to obstacle (m)
R1,R2,R3 = zones
floor/roof is relative to R2/R1 (0,1)
floor/roof-gap has interval (0, 0.5)
r1size is messured in meters

Figure 1: For the 2D occupancy grid to work, each obstacle sensor must specify a sensor model.

```
   // Do something with the speed value
}

// when done (e.g in close()), remove the observer:
speedSensor.deleteSpeedObserver(speedObserver);
-----------------------------------------------------
```

To fuse data from several obstacle sensors (RangeArraySensor), a 2D occupancy grid, is also available. This class, called *Map2D,* is located in the package com.ifor.sensor. To get all obstacles within a specified radius, the method *getObstaclesWithinRadius* is used (the radius can be specified in Map2D's ini-file). For the occupancy grid to work, each RangeArraySensor must specify a sensor model, as seen in Figure 1. As with sensors, each model has its own ini-file specifying the variables, and is loaded by the cfg-file.

4 Implementation

The following sample code first starts the Nav2000 system with a cfg-file and get references to all sensors involved. Then open is called, and after the system has been set in auto-mode, a speed is set and then read after a short pause. Last but not least, the system is closed and then we exit completely from Nav2000.

```
----------- Some initilization -----------
String cfgFile = "C:/IFOR/CVS_checkout/ifor2_head/cm_ini/Pioneer/amigobot";
n2k = new NAV2000(cfgFile);
```

```
speedSensor = (AriaSpeedSensor) Registry.lookUp("pioneerspeedsensor");
angleSensor = (AngleSensor) Registry.lookUp("pioneeranglesensor");
headingSensor = (HeadingSensor) Registry.lookUp("pioneerheadingsensor");
positionSensor = (PositionSensor) Registry.lookUp("pioneerpositionsensor");
sonar = (RangeArraySensor) Registry.lookUp("pioneersonararray");
vehicle = (AriaVehicle) Registry.lookUp("vehicle");

----------- Move the robot a bit -----------
n2k.open(); // start all sensors
vehicle.setAuto(); // we must be in auto mode
vehicle.setSpeed(0.3, -0.25);
Thread.sleep(1000); // wait for 1000 ms
System.out.println("Speed: " + speedSensor.getSpeed());
Thread.sleep(3000); // wait for 3000 ms
n2k.close(); // close all sensors
n2k.quit(); // exit NAV2000
```

5 Coordinate system

To use several sensors, a uniform coordinate system must be defined for all sensors and the vehicle. The global coordinate system is the one that the position and heading sensors work in. This system is fixed on the ground and defined with x-axis to the east and y-axis to the north. In addition to the global coordinate system, there is a local one fixed to the robot. All sensor poses (how the sensors are mounted) are defined relative to this system. We want all sensor readings in the local pose to be expressed in the global coordinate system, refer to Figure 2. Obstacle sensors work in a different coordinate system, as they give a range and direction to an obstacle relative to the sensor. These readings are then converted either to the local or global coordinate system depending on what application is using them. For the sake of completeness and clarity, the properties of the different coordinate systems as well as related definitions are listed below.

Global coordinate system

- The (x, y, z) coordinate system is right-handed
  - The origin is arbitrarily defined and is different for different sensors. We use the GPS coordinate system as reference.
  - The (x, y) plane is defined as horizontal (i.e. perpendicular to the gravitational field)
  - The x axis points eastward
  - The y axis points towards the geographic north pole
  - The z axis points upwards from the (x, y) plane

Local coordinate system

- The origin is an arbitrarily chosen point in the robot (usally in the middle)
- The (x', y', z') coordinate system is right-handed
- The (x', y') plane is defined as horizontal on flat ground
- The x' axis points forward along the robot
- The y' axis points to the left
- The z' axis points upwards

The local pose expressed in the global coordinate system

- Position $= (x, y, z)$
- Attitude $=$ (roll, pitch, yaw) $= (\phi, \theta, \psi)$
- Pose $=$ position and attitude $= (x, y, z, \phi, \theta, \psi)$

Important angles The signs of these angles are determined by the right-hand rule. Refer to Figure 2.

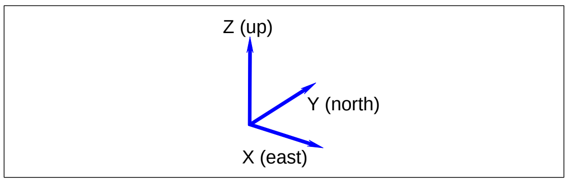
- Roll (a.k.a. bank angle) ϕ : the angle between y' and the (x, y) plane. $(-\pi < \phi \leq \pi)$
- Pitch (a.k.a. elevation) θ : the angle between x' and the (x, y) plane. $(-\pi < \theta \leq \pi)$
- Yaw (a.k.a. heading or azimuth) ψ : the angle between x and the projection of x' on the (x, y) plane. $(-\pi < \psi \leq \pi)$

References

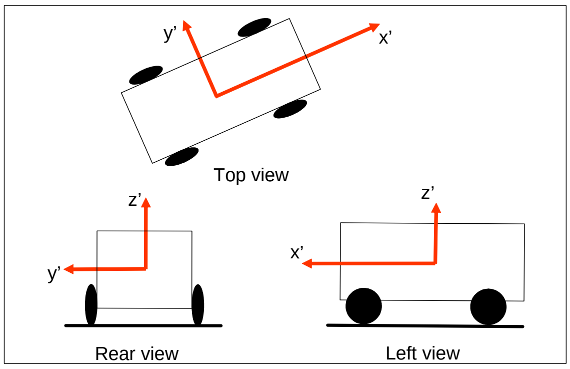
[1] Thomas Hellström, Thomas Johansson, and Ola Ringdahl. A software framework for control and sensing in mobile robotics. Technical Report UMINF 07.05, Department of Computing Science, Umeå University, April 2007.

[2] Thomas Johansson and Thomas Hellström. A software infrastructure for sensors, actuators, and communication. In *In Proceedings of The Third Swedish Workshop on Autonomous Robotics (SWAR05)*, 2005.

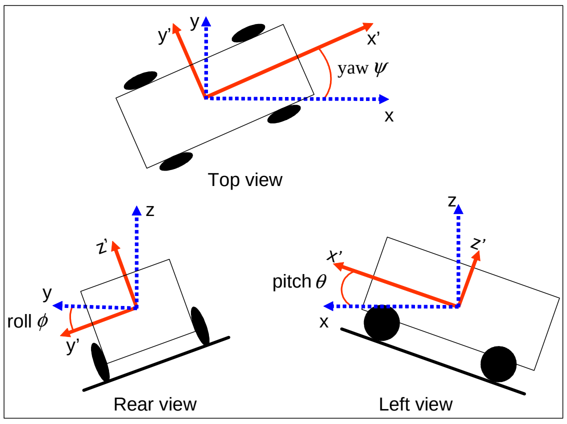
[3] Ola Ringdahl. *Techniques and Algorithms for Autonomous Vehicles in Forest Environment.* Licentiate thesis, Department of Computing Science, Umeå University, 2007.



(a) Global coordinate system (x,y,z)



(b) Local coordinate system (x',y',z')



(c) Local pose expressed in the global coordinate system

Figure 2: Definition of the coordinate system.